AFOSR-TR- 88-1028

THE BDM CORPORATION

7915 Jones Branch Drive, McLean, Virginia 22102-3396 • (703) 848-5000 • Telex: 901103 BDM MCLN

Optics and Symbolic Computing Annual Technical Report

PREPARED FOR AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFOSR)
AND DEFENSE ADVANCED RESEARCH PROJECTS AGENCY (DARPA)

APRIL 1988

BDM/MCL-88-0074-TR

SECURITY CLASSIFICATION OF THIS PAGE

ADA200564

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	Approved for public release; distribution unlimited.
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 88-1026
6a. NAME OF PERFORMING ORGANIZATION: BDM Corporation	6b. OFFICE SYMBOL (If applicable):
7a. NAME OF MONITORING ORGANIZATION: AFOSR/NE	
6c. ADDRESS (City, State and ZIP Code): 7915 Jones Branch Drive, McLean, VA 22102-3396	7b. ADDRESS (City, State and ZIP Code): Bldg 410, Bolling AFB DC 20332-6448
8a. NAME OF FUNDING/SPONSORING ORGANIZATION: AFOSR	8b. OFFICE SYMBOL (If applicable): NE
9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER: F49620-86-C-0030	
8c. ADDRESS (City, State and ZIP Code): Bldg 410, Bolling AFB DC 20332-6448	10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO.
61102F	DARPA		

11. TITLE (Include Security Classification): Optics and Symbolic Computing

12. PERSONAL AUTHOR(S): Athale

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Annual Report	FROM _____ TO _____		

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB. GR.	

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

During the period covered by this report, two new efforts were undertaken that demonstrate how optical techniques can alleviate specific communications and processing bottlenecks in symbolic computation. In particular, the first effort developed the folded perfect shuffle optical processor to serve as the communication hardward for parallel processors intended to speed up the solution of AI problems. The second effort addressed the application of parallel optical Boolean matrix operations to prune the search space of AI problems that are represented by the consistent labeling formuulation. In the following paragraphs, the salient features and impact of these two efforts will be discussed. The next two sections of this report contain journal articles that describe these efforts in more details.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT: UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS ☐	21 ABSTRACT SECURITY CLASSIFICATION: UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Giles	(202) 767-4931	NE

DD FORM 1473, 83 APR — EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

THE BDM CORPORATION

7915 Jones Branch Drive, McLean, Virginia 22102-3396 • (703) 848-5000 • Telex: 901103 BDM MCLN

OPTICS AND SYMBOLIC COMPUTING
ANNUAL TECHNICAL REPORT
April 1988

BDM/MCL-88-0074-TR

Prepared for Air Force Office of Scientific Research (AFOSR) and Defense Advanced Research Projects Agency (DARPA).

TABLE OF CONTENTS

LIST OF FIGURES

APPLICATION OF OPTICS TO PROBLEMS IN SYMBOLIC COMPUTATION

ANNUAL TECHNICAL REPORT - JANUARY 1988

INTRODUCTION

The goal of the program "Application of Optics to Problems in Symbolic Computation" at BDM is to examine the impact that optical computing may have on outstanding problems in AI. During the period covered by this report, two new efforts were undertaken that demonstrate how optical techniques can alleviate specific communication and processing bottlenecks in symbolic computation. In particular, the first effort developed the folded perfect shuffle optical processor to serve as the communication hardware for parallel processors intended to speed up the solution of AI problems. The second effort addressed the application of parallel optical Boolean matrix operations to prune the search space of AI problems that are represented by the consistent labeling formulation. In the following paragraphs, the salient features and impact of these two efforts will be discussed. The next two sections of this report contain journal articles that describe these efforts in more detail.

The perfect shuffle (PS) is an important interconnection pattern in parallel processing. It has been shown to be capable of speeding up computation of the FFT and matrix operations along with routing messages in a centralized or distributed manner for parallel computer architectures. However 2D implementations of the PS in either VLSI or printed circuit board technology, must consume at least $O(N^2/\log^2 N)$ surface area. For large N, the amount of area that the network requires can be prohibit single chip or single board implementations. Thus, large PS's must cross at least one level of the computer organization heirarchy, seriously degrading the network performance in terms of physical size, power consumption, and signal delay. Another limitation of 2D technologies is as the chip area grows, for the area-optimal layouts so does difference in lengths of the longest and shortest communication paths. This implies that there will be a size-dependent signal skew which limits the signal bandwidths of synchronous systems.

To alleviate the performance limitations imposed by 2D electronic technology, BDM developed the folded perfect shuffle optical processor. The folding strategy begins by raster encoding the 1D list into a 2D array. The shuffling operations are also transformed into 2D and applied to the 2D-formatted data. This approach uses the 2D formating and 3D connection capabilities of optics to perform large perfect shuffles that require only O(N) area; allow the per channel power to be independent of N; have low delay; and eliminate signal skew, allowing high bandwidths. These properties will allow future parallel processors to be larger and to operate at higher speeds, significantly reducing the time it takes parallel processors to solve computationally intensive AI problems. The details of our architecture are given in the attached reprint of a paper published recently in *Applied Optics*.

The second task undertaked during this reporting period was an investigation of optical techniques for application to search problems in Artificial Intelligence (AI). The tree search or graph matching problem is ubiquitous in AI. Applications areas include: scheduling, theorem proving, and scene labeling/interpretation for computer vision. In general these problems have exponential time complexity and become intractable rapidly as the number of variables grows. A large body of research has been dedicated to developing "tree-pruning" techniques, which use forward checking to increase the efficiency of the search. These techniques attempt to avert the combinatorial explosion by using the relational constraints of the problem in local graph operations (arc and path consistency checks) to reduce the complexity of the search tree. Under worst case assumptions, forward checking itself requires exponential time; however, for many real world problems, it does increase the efficiency of the search.

A tree search can be formulated as a *consistent labeling* (CL) problem, in which the goal is to assign a *label*, from a set of L elements, to each *unit*, from a set of U elements. U corresponds to the number of levels in the search tree and L corresponds to the number of branches at each node of the tree. Not all of the L^U possible assignments are permited by the problem constraints and the goal of a forward checking algorithm is to rule out, in advance, those partial labelings which cannot possibly contribute to a CL, where a CL is defined as a labeling of all U units in which all of the labelings are simultaneously compatible with the problem constraints.

The initial problem constraints are given as tuples of units which mutually constrain each other, along with the sets of allowed labels for each tuple. Here we restrict our attention to binary constraints. Many interesting problems in the application areas mentioned above can be cast as CL problems with binary constraints. For such problems the constraint data can be represented as LxL Boolean matrices, one for each pair of units that constrain each other.

In this task we investigated the potential for improving the efficiency of the search by applying highly parallel optical Boolean matrix operations to the set of constraint matrices. The purpose of these operations is two-fold. First, we want to remove, from the initial set of binary constraints, as many as possible of those that do not contribute to any consistent labeling. This improves the efficiency of the search by reducing the size of the domain of allowed pair labelings that must be checked during the search procedure. The second purpose in manipulating the constraint matrices is to make explicit those unary constraints that are implied by the initial set of binary constraints. These *induced* unary constraints can then be applied directly in the search process to prune the search tree. Our ideas are detailed in a preprint of a paper attached to this report which has also been submitted to the *Optical Engineering* special issue on optical computing.

Reprinted from Applied Optics, Vol. 27 page 202, January 15, 1988

Folded perfect shuffle optical processor

Charles W. Stirk, Ravindra A. Athale, and Michael W. Haney

BDM Corporation, 7915 Jones Branch Drive, McLean, Virginia 22102.
Received 25 September 1987.
Sponsored by J. W. Goodman, Stanford University.
0003-6935/88/020202-02$02.00/0.

The perfect shuffle interconnection network (PS) consists of splitting a linear array of $N = 2^n$ items in half and interleaving the two halves. The PS was originally proposed as the interconnection primitive between local processors for parallel computation of the fast Fourier transform polynomial evaluation, sorting, and matrix transposition.[1] Interconnection permutations necessary for parallel matrix computations other than matrix transpose also were realized on the PS.[2] For the more general routing problems found in MIMD machines and telecommunications networks, an algorithm was presented to connect an arbitrary permutation of inputs to outputs with a limited number of PS stages $[O(\log N)]$.[3] A review of the parallel computation abilities of the PS is conducted in Ref. 4.

The wide utility of PS networks led to attempts to map the PS onto VLSI architectures.[5,6] The inherent VLSI wiring characteristics of capacitive and inductive crosstalk, length-dependent power requirements, timing skew, limited crossovers, and chip area are ill-suited to the global and space-variant nature of the PS. Thus a trade-off of local processing complexity, the number of parallel channels and signal bandwidth due to the practical limitations of VLSI, limits the applicability of electronic PS implementations.

The ability of light beams to carry high-bandwidth data through free space without mutual coupling, the independence of drive power and interconnection length, the minimal timing skew, and the 3-D nature of optical systems led to suggestions of optical interconnects for electronic processors.[7] The space-bandwidth product of optical systems (10^6) limits the number of parallel channels, while the modulation speed (10 GHz) and degree of multiplexing (wavelength and polarization) limit the channel bandwidth. By using optical communications the overall system throughput is increased by eliminating the interconnection network bottleneck. Hence free-space optics appears ideal for implementation of the PS in applications with high data rates or many parallel channels like telecommunications and fine-grained parallel processors.

The earliest proposed free-space optical architectures for the PS[8,9] are capable of shuffling either the rows or columns of a matrix. If each data word is recorded as a column, shuffling between the columns results in a bit-parallel PS. In such spatially multiplexed architectures the maximum length of the data words must be known *a priori*. With time-multiplexed data streams, however, the word length is unlimited, and the other spatial dimension is free to encode additional data channels. Moreover, in many cases, the subsystems connected by the PS provide and require serial data. A 2-D optical PS was proposed[10] and demonstrated[11] that rearranges a 2-D image of serial data streams by shuffling the rows and columns. The architecture divides and interlaces the matrix along each direction as separate operations. Since the 1- and 2-D PS are different interconnection primitives, not all the routing algorithms and hardware developed for the 1-D PS are compatible with the 2-D PS.

In this Communication we describe an algorithm that performs a 1-D PS on a long 1-D sequence that is raster-formatted on a matrix. We propose optical architectures and hardware to implement the algorithm and show experimental results for a particular system. The proposed algorithm and architectures retain compatibility with the well-developed 1-D PS algorithms while taking full advantage of the 3-D interconnection capabilities and 2-D space–bandwidth of free-space optics. Because a similar philosophy was employed in designing 2-D optical systems to perform spectrum analysis of very long (10^6) raster-recorded 1-D time signals in the work on the folded spectrum analyzer,[12,13] we call our processor the folded perfect shuffle optical processor.

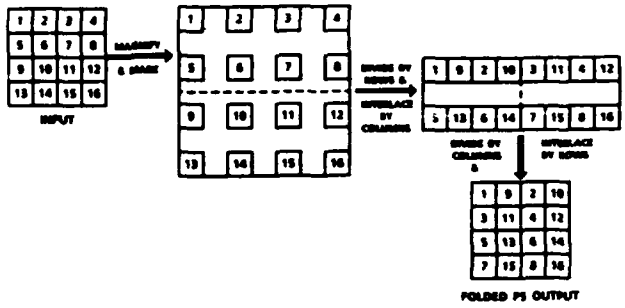


Fig. 1. Two-step approach to the folded PS.

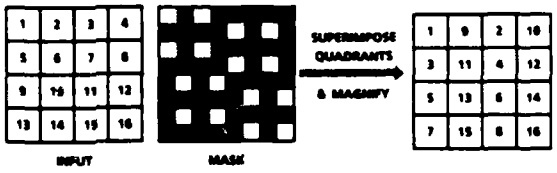


Fig. 2. One-step approach to the folded PS.

The 1-D PS algorithm is shown in Eq. (1) where N is the number of channels, i is the original data index, and i' is the shuffled index. (All indices start at 0.) Since the operations on the indices are linear and commute, they can be performed in any order or simultaneously:

$$\begin{aligned} &\text{if } 0 < i < N/2 - 1, \text{ then } i' = 2i; \\ &\text{if } N/2 < i < N - 1, \text{ then } i' = 2i + 1 - N. \end{aligned} \tag{1}$$

The details of the 1-D and 1-D folded PS algorithms differ because the data formats are fundamentally dissimilar. While the linear algorithm (and 2-D) requires the list (image) to be divided and interlaced along the same direction (both directions independently), in the folded algorithm the matrix is divided in half along the rows and interlaced along the columns. This results in a shuffled version of the input, but the rows are now twice as long and the columns half as high as those in the input matrix. Hence the output matrix must be further divided along the columns and interlaced along the rows for input/output compatibility (Fig. 1). Interchanging row and column operations in the preceding sequence produces the same folded output. These operations can also be performed simultaneously by dividing, magnifying, shifting, and interlacing the quadrants (Fig. 2). Masking is necessary for the area of the output pixels to be compatible with the input. In some architectures masking prevents pixel overlap.

Dividing a raster encoded matrix along both directions mandates that the dimensions be even. Optically dividing a matrix is possible using beam splitters, mirrors, Wolloston prisms with polarization encoding, gratings, or lenslet arrays. Addition of a constant, minus N or plus 1, is a simple shift of a quadrant in the verticle or horizontal directions, respectively. Practical shifting hardware includes tilted mirrors, off-axis lenses or lenslet arrays, prisms, or gratings. Multiplying the position by two involves imaging with 2× magnification. Figure 3 depicts a one-step architecture for the folded perfect shuffle. A simple mask at the input plane recodes the image for magnification and shifting. The four imaging lenses perform the magnification and shifting by overlapping the quadrants at the output plane. Due to the input recoding, the overlapped quadrants produce the desired PS output. The experimental results for 64-point perfect shuffles are shown in Fig. 4. The architecture is easily extendable to larger numbers of parallel channels.

In summary: We desire a folded perfect shuffle optical processor for several reasons. First, the 1-D perfect shuffle is an important interconnection primitive in communications and parallel computation. Second, formatting the 1-D data channels in two dimensions efficiently uses the 3-D interconnection capability of optics for potentially upward of 10^6 parallel channels. Moreover, parallel channels and serial data are compatible with the requirements of many preceding and subsequent information handling subsystems. Finally, the passive nature of the optical architectures presented allows shuffling times of the order of a nanosecond and are readily pipelinable. In this Communication, we described an algorithm that performs a folded perfect shuffle. We outlined the architectural and hardware approaches to implement the algorithm optically and demonstrated a particular architecture on a 64-point perfect shuffle.

This research was supported by the Advanced Research Project Agency of the Department of Defence and was monitored by the Air Force Office of Scientific Research under contract F-49620-86-C-0030.

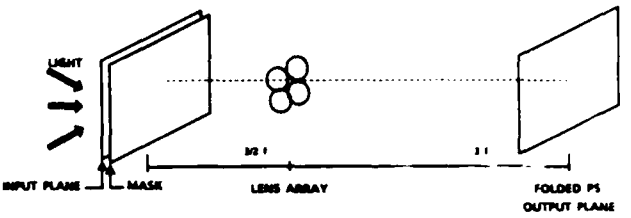


Fig. 3. One-step imaging architecture for the folded PS.

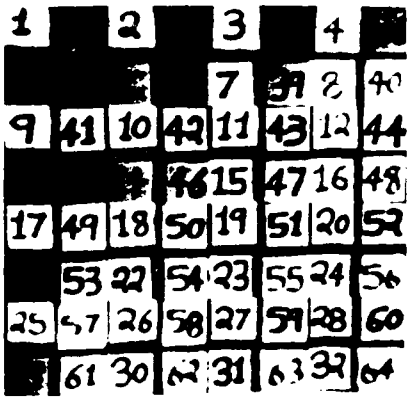


Fig. 4. Experimental results for sixty-four-channel folded PS. The variation in contrast is due to the nonuniformities in the input illumination.

References

1. H. S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Trans. Comput. **C-20,** 153 (1971).
2. D. H. Lawrie, "Access and Alignment of Data in an Array Processor," IEEE Trans. Comput. **C-24,** 1145 (1975).
3. C. L. Wu and T.-Y. Feng, "The University of the Shuffle-Exchange Network," IEEE Trans. Comput. **C-30,** 324 (1981).
4. J. T. Schwartz, "Ultracomputers," ACM Trans. Program. Lang. Syst. **2,** 484 (1980).
5. F. T. Leighton, *Complexity Issues in VLSI: Optimal Layouts for the Shuffle-Exchange Graph and Other Networks* (MIT Press, Cambridge, 1983).
6. C. D. Thompson, "The VLSI Complexity of Sorting," IEEE Trans. Comput. **C-32,** 1171 (1983).
7. J. W. Goodman, F. J. Leonberger, S.-Y. Kung, and R. A. Athale, "Optical Interconnections for VLSI Systems," Proc. IEEE **72,** 850 (1984).
8. A. Lohmann, W. Stork, and G. Stucke, "Optical Implementation of the Perfect Shuffle," in *Technical Digest, Topical Meeting on Optical Computing* (Optical Society of America, Washington, DC, 1985), paper WA3.
9. J. E. Midwinter, " 'Light' Electronics, Myth or Reality?," IEE Proc **132,** Pt. J, No. 6, 371 (1985).
10. A. Lohmann, "What Classical Optics can do for the Digital Optical Computer," Appl. Opt. **25,** 1543 (1985).
11. S.-H. Lin, T. F. Krile, and J. F. Walkup, "2-D Optical Multistage Interconnection Networks," Proc. Soc. Photo-Opt. Instrum. Eng. **752,** 209 (1987).
12. C. E. Thomas, "Optical Spectrum Analysis of Large Space Bandwidth Signals," Appl. Opt. **5,** 1782 (1966).
13. T. M. Turpin, "Spectrum Analysis Using Optical Processing," Proc. IEEE **69,** 79 (1981).

Optical Techniques for Increasing the Efficiency of Heuristic Search

Michael W. Haney, Ravindra A. Athale, and Roger A. Geesey

BDM Corporation

7915 Jones Branch Drive

McLean, VA, 22102

ABSTRACT

Many problems in Artificial Intelligence are intractable due to the exponential growth of the solution space with problem size. Often these problems can benefit from heuristic search or forward-checking techniques which attempt to prune the search space down to a managable size before or during the actual search procedure. Many interesting search problems can be formulated as consistent labeling problems in which the initial problem information is given in the form of a set of binary constraints, for which Boolean matrices are a natural data represention. In this paper optical implementations of Boolean matrix operations are proposed for manipulating the constraint matrices to perform forward-checking and thereby increase the search efficiency. The high degree of parallelism afforded by using optical techniques and the relatively low accuracy requirements of Boolean matrix operations suggest that optical techniques are well matched to this problem.

I. Introduction/Background

Problems that require searching through very large solution spaces are ubiquitous in Artificial Intelligence (AI). Examples can be found in: expert systems, scheduling, theorem proving, database management, game playing, decoding, and computer vision. In general these problems have exponential computational complexity and solutions based on exhaustive search are impractical. These problems are characterized by a lack of structure in the solution space which precludes algorithmic solutions. Heuristic search strategies are therefore envisioned as the only hope for attacking these problems.

Many AI problems can be formulated as consistent labeling (CL) problems [1], in which the goal is to assign a label, from a set of L possible labels, to each unit, from a set of U units, while satisfying all the known constraints of the problem. The initial problem constraints are given as a set T of n-tuples of units which mutually constrain each other, along with a set R of 2n-tuples which list the allowed label assignments for each n-tuple in T. The constraint relations defined by T and R can, in general, consist of unary-, binary-, ternary-, ...- relations, depending on the number of elements in the n-tuples.

To make these notions less abstract, it is appropriate to describe the CL formulation in terms of a real-world example. Consider the problem of scheduling N speakers (denoted: a,b,c,...) into N timeslots (denoted: 1,2,3,...,N). The set of units corresponds to the various timeslots and the set of possible labels for those timeslots corresponds to the set of

speakers. An example of a given unary constraint is: "Speaker 'a' is unavailable for timeslots 20 through 30". In the CL formulation this corresponds to having the set of individual units being self-constraining as 1-tuple members of the set T. The corresponding members of the set R are of the form: (1,a), (2,a),...,(19,a), (31,a),..,(N,a). This set corresponds to the allowed label assignments taking into account the known unary constraint. An example of a binary constraint is: "Speakers 'b' and 'c' should not be scheduled back-to-back". The corresponding members of the set T are then all pairs of units (timeslots) that are in sequence, i.e., (1,2), (2,3),...,(N-1,N), and the corresponding members of the set R are of the form: (1,i;2,j),(2,i;3,j),...,(N-1,i;N,j), such that i and j are not simultaneously b and c. An example of a possible ternary constraint is: "Speakers 'a','e', and 'f' should be scheduled in sequence;" the appropriate members of T and R can be listed in an analogus manner to the binary constraints.

In this paper we restrict our attention to binary constraints -- the units directly constrain each other only in a pairwise manner. Many interesting problems in the application areas mentioned above can be cast as CL problems with binary constraints. Examples of such problems include the Satisfiability problem, which is an archtypical NP-complete problem related to theorem proving, and scene labeling in computer vision, among others [3].

II. Consistent Labeling Problem with Binary Constraints

1. Graphs and Tree Structures

When cast as a directed graph problem, the nodes of the graph correspond to the units and the connecting arcs of the graph correspond to relations between the nodes that define the allowed labels for each node. The search process consists of moving from node to node in the graph, assigning labels to each unit (node), and checking to determine whether the label assignments made so far are consistent with the given problem constraints.

Figure 1 illustrates an example of a binary constraint search problem involving 5 units and 3 labels. The initial set of pairs of units which constrain each other is given in set T and the allowed set of labels for these constraining pairs is given in the set R. Since only binary constraints are considered, the graph representation in Figure 1 has arcs connecting nodes in a pairwise manner. Furthermore, we assume for this simple example that the order of the units in the constraining pairs does not matter. In this case the graph representation is an undirected graph as shown.

The recursive nature of the graph search process is often represented as a tree structure to permit the time history of the search process to be preserved. For the CL problem formulation, U, the number of units, corresponds to the number of levels in the search tree, and L, the number of candidate labels, corresponds to the number of branches at each node of the tree. Not all of the L^U possible assignments are permited by the

problem constraints and the goal of the search is to find which (if any) of the possible labelings are all simultaneously compatible with the problem constraints. Such a labeling is called a Consistent Labeling. The initial search tree associated with the problem depicted in Figure 1 is shown in Figure 2. It can be seen that even such a simple problem has $L^U = 243$ possible paths that might need to be checked for a consistent labeling in a brute-force search, revealing the combinatorial explosion.

In a standard back-tracking tree search procedure a trial label is assigned to the first level (unit) of the tree and checked to see if it is consistent with the given constraints. If the label is not consistent a new label is tried until one is found that is allowed. When a permitted label is found, a trial label is assigned to the second level of the tree, and the trial pair of labelings is checked for consistency with the given constraints. This procedure is maintained down through succesive levels of the tree until either a CL (i.e., an allowed path) is found or a unit is discovered for which no allowed label can be found. In the latter case the procedure must back up one level of the tree (i.e., to the previous unit) and find another consistent label before continuing. If no label can be found at that level then the search must back up one more level until an allowed labeling is found that is different from the one that caused the procedure to halt and backtrack. If the search is forced to backtrack all the way to the first level without finding a CL then no CL exists. The problem with this procedure

is that it suffers from "thrashing" behavior [2] -- a trial labeling at a high level in the tree which is not part of a CL may not be discovered without checking down many paths and backtracking many times.

A large body of research has been dedicated to developing "tree-pruning" techniques, which use forward checking to forstall thrashing and increase the efficiency of the search. These techniques attempt to avert the combinatorial explosion by using the relational constraints of the problem in local graph operations (arc and path consistency checks) to reduce the complexity of the search tree. Under worst case assumptions, forward checking itself requires exponential time; however, for many real world problems, it does increase the efficiency of the search [3].

2. Data Representation

For problems with binary constraints the relation data can be represented as LxL Boolean matrices, R(i,j), one for each pair of units (i,j) that constrain each other [2]. The L rows correspond to the labels of unit i and the L columns correspond to the labels of unit j. The presence of a "1" as the kl^{th} element of matrix R(i,j) indicates that assigning the i^{th} unit with label "k" is consistent with labeling the j^{th} unit "l". Note that if two units are not given to constrain each other directly, then the initial constraint matrix corresponding to that pair would consists of all 1's and contains no useful information about how that pair of units might ultimately constrain each other through induced constraints. Figure 3 shows

the seven binary constraint matrices corresponding to the given allowed labelings in the example problem of Figure 1. Since the ordering of the constraining pairs is not important in this problem it is noted that $R(i,j) = R^T(j,i)$, where ^{T} indicates transpose of the matrix.

3. Forward-checking

In general, the set of given constraints is very large. For those search problems for which there are only a small number of possible CLs, this means that many of the initially allowed label assignments are superfluous -- they do not contribute to any of the CLs. Indeed, a large portion of the inefficiency in brute-force search strategies can be attributed to the repeated checks of these extraneous contraint relations during the search. An important objective of a forward-looking procedure is therefore to discard as many as possible of those constraints that play no role in a final solution [1], thereby avoiding the inefficiency of repeatedly checking them during the search.

Various levels of forward-checking can be achieved, corrsponding to looking ahead to various levels in the search tree. An arc of the graph is consistent if the set of allowed labelings for the pair of nodes (units) connected by the arc is also consistent with the allowed labelings of all other pairs of nodes that contain one of the two initial nodes. If all arcs of the graph are consistent, then the arc consistent network [2] has been computed, i.e., we have looked-ahead to all paths of length 1 from each node of the graph. This concept can be generalized to paths of length 2 and higher [2].

In the next section we discuss the application of highly parallel optical Boolean matrix operations to the set of constraint matrices to perform arc and path consistency checks. The purpose of these operations is two-fold. First, as stated earlier, we want to remove, from the initial set of binary constraints, as many as possible of those constraints that do not contribute to any consistent labeling. This improves the efficiency of the search by reducing the size of the domain of allowed pair labelings that must be checked during the search procedure. The second purpose in manipulating the constraint matrices in forward checking operations is to make explicit those unary constraints (i.e., unit u_i cannot assume label l_k) that are implied by the initial set of binary constraints. The derivation of a unary constraint from the given binary constraints constitutes a significant reduction in the size of the search space. Each induced unary constraint for a unit u reduces the size of the search space by a factor of $1-L_u/L$, where L_u is the current number of labels for unit u found to be not permitted by induced unary constraints, and L is the total number of labels. The induced unary constraints can thus be applied directly to the search tree.

As the forward-checking process proceeds, the constraint matrices become more sparse due to the deletion of extraneous constraints. Eventually, the sparsity of the matrices may become such that an entire row or column contains only zeros. This situation indicates an induced unary constraint. The operation of unary constraint detection is accomplished by examining each

of the current set of binary constraint matrices for the presence of a row or column containing only 0's. This situation indicates that the unit associated with the rows or columns of that matrix can never be assigned the label corresponding to that row or column. This can be detected by performing an OR operation across all rows, and then all columns, for each of the constraint matrices. If an all-zero row or column is found, then the resulting induced unary constraint can be propagated to all other contraint matrices that share the same unit by zeroing out the associated row or column associated with that label and unit. This may lead to the discovery of new induced unary constraints which can be detected and propagated until a fixed point is reached.

The first step in a breadth-first forward-checking procedure is to check for unary constraints that might be already present in the initial binary constraints. Examination of the matrices for the sample problem in Figure 3 reveals that R(2,3) has a row and a column that consists of all zeros. The all-zero row corresponds to the unary constraint: unit 2 cannot have label b, and the all-zero column corresponds to the unary constraint: unit 3 cannot have label c. Propagation of these unary constraints results in the zeroing out of the third column of R(1,3) and the third row of R(3,4). The new set of reduced constraint matrices is shown in Figure 4. Applying the unary constraints to the search tree reduces the number of possible paths that might possiblly need to be checked to 108 as shown in Figure 5.

The detection and propagation of all unary constraints hidden in the initial set of binary constraints corresponds to the computation of the arc consistent network for the problem. To achieve further benefit from forward-checking requires the use of path consistency checks for paths of length 2 or greater.

III. Optical Matrix Manipulations for Pruning the Search Tree

1. Boolean Matrix Operations

The Boolean matrix operations of intersection and composition have previously been suggested for use in forward checking [2]. Intersection is the element by element ANDing of corresponding entries of two matrices to yield a new matrix. Composition is a Boolean matrix multiplication which is obtained by replacing the multiplication and addition operations in conventional matrix multiplication by Boolean AND and OR operations, respectively.Since the multiplication between two binary elements is equivalent to their multiplication, the only oprerational difference is in the generation of binary output elements by replacing the analog addition with multi-input OR operation. Here we propose that these operations can be combined with unary constraint detection and unary constraint propagation, to compute path consistent networks which will increase the efficiency of the search. Furthermore, for increased speed, all of these operations can be implemented optically, using established low accuracy analog linear algebraic techniques, followed by simple nonlinearities at the detector stage to implement the OR operation.

The use of intersection and composition in a forward checking operation proceeds as follows. Given a constraint matrix relating units i and j, R(i,j), and other constraint matrices R(i,k) and R(k,j), we can create a new constraint matrix:

$$R'(i,j) = R(i,j)*R(i,k)\&R(k,j), \quad (1)$$

where "*" indicates the intersection operation and "&" indicates the composition operation. Composition takes precedence over intersection. The induced relation R' replaces R and is a stronger constraint between units i and j because it now takes into account the influence of an intermediate unit (k) along the path and not just the single arc between the units. Even stronger constraints can be derived by intersecting R'(i,j) with other induced constraint matrices derived from the composition of matrices along other paths between i and j. In practice, this operation would be performed on all constraining pairs of units to some level of path length.

In the example problem, the application of Equation 1 to compute new versions of R(1,4) and R(1,5), as shown in Equations 2 and 3:

$$R'(1,4) = R(1,4)*R(1,3)\&R(3,4), \quad (2)$$

$$R'(1,5) = R(1,5)*R(1,4)\&R(4,5), \quad (3)$$

These path consistency checks yield new unary constraints which, after propagtion, result in the set of constraint matrices shown in Figure 6. With the new induced unary constraints (unit 1 cannot be labeled "b", unit 5 cannot be labeled "a") the search tree is now pruned such that only 48 paths remain to be checked in a search procedure, as shown in Figure 7. A check of these paths shows that there are 3 CLs for the example problem: (1a,2a,3a,4b,5b), (1a,2c,3a,4b,5b), and (1c,2c,3b,4a,5c).

2. Optical Implementation

Optical implementation of the operations of intersection and composition has previously been suggested in the context of inference machines [5]. The operation of intersection can be implemented optically via image multiplication by representing the constraint matrices as binary images, while composition can be implemented by analog optical matrix multiplication, followed by thresholding to restore the levels to 1 or 0.

Unary constraint detection is achieved by focusing the light passed through the matrices along each dimension separately and detecting the presence of light with a 1-D threshold detector array. This achieves the required OR operation across all rows or columns simultaneously. To propagate the induced unary constraint the resulting thresholded 1-D data, R(i), is transformed into a light signal, spread out into a 2-D array, and multiplied by all other matrices, R(i,m), which share the unit that has the unary constraint. For constraint matrices which involve the unit i as the column index, the transpose, $R^T(m,i)$ is used.

IV. Discussion

The forward-checking process could conceivably continue until all the superfluous binary constraints and resulting unary constraints are discovered. At this point the minimal network [2] has been computed, which means that every remaining binary labeling in the constraint relations is a part of at least one CL and forward checking will not reduce the set of binary constraints any further. It has been shown that the closer a network is to minimal then the less work the tree search will have [3]. However, computation of the minimal network for a general combinatorial problem is itself an NP-complete problem [4]. In fact forward-checking is not guaranteed to reduce the computational requirements of the search. However, some problems have proven to benefit greatly from the application of forward-checking [2].

Rather than carrying out the entire search procedure using optical forward-checking techniques, it is envisioned that the application of optical techniques for forward-checking is appropriate as a pre-filtering operation to remove as many of the superfluous binary constraints as possible and prune the search tree down as much as possible before turning the problem over to a conventional Prolog-type processor to complete the search. In this context, optical techniques would be applied to those computationally intensive forward-checking operations of the search that are amenable to highly parallel optical techniques, while the control intensive and sequencial operations of the

search would be left to the more conventional electronic techniques.

The extent to which optical forward-checking techniques should be employed will depend on the size and nature of the problem and the heuristic search strategy used. Current conventional wisdom assumes that Boolean constraint matrices as large as 1000 x 1000 pixels can be handled optically. This would seem to suggest that an upper limit on the number of labels in the problem would be on the order of 1000. However, this limit may not be correct -- the large matrix operations can be decomposed into a set of smaller ones without loss of computational efficiency. The number of units that can be handled with optical techniques appears to be limited principally by the amount of memory needed to store the large number of constraint matrices -- the more units in a problem, then the more likely a binary constraint relation exists between a pair of units.

The number of unary constraints that can be detected and propagated, and hence the importance of these operations to the given problem, is determined by the number of CLs that are possible (something you may not have a handle on ahead of time) and the number of labels and units for the problem. Clearly if only one CL exists then the number of unary constraints that may be detected through forward-checking operations grows with the size of the sets of units and initial candidate labels. At the other extreme, if the number of CLs is large and the number of candidate labels is low, then it is possible for there to be no

unary constraints in the problem, which simply means that each units is allowed to be assigned each of the labels for at least one CL.

Although optical implementations of the Boolean matrix operations and manipulations on the set of constraint matrices offer the potential for great speed due to the parallel nature of the computations, the overall speed of a hybrid architecture may be limited due to bottlenecks associated with the required control and memory manipulation functions. Effective use of optical matrix operations may require an architecture which has a very limited amount of transductions between the optical and electronic domains to avoid the associated bottlenecks. In fact, the best approaches may keep the constraint data in an optical matrix format throughout the forward-checking procedure, and return to the electronic domain only during the actual search process. The ultimate utility of optical forward-checking techniques will be determined by the progress made in optical mass storage technology.

V. Acknowledgments

This work was supported by DARPA/DSO and monitored by Air Force Office of Scientific Research.

VI. References

[1] Haralick, R. M. and Shapiro, L. G., IEEE Trans. PAMI, Vol. PAMI-2. No. 3, May 1980, pp. 193-203.

[2] Mackworth, A. K., Artificial Intelligence 8 (1977), pp. 99-118.

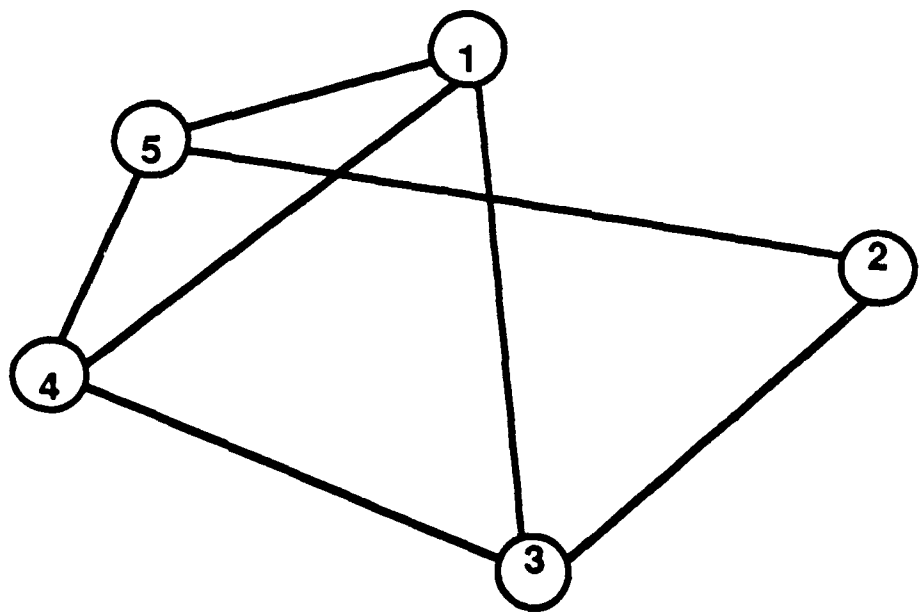
[3] Haralick, R. M. and Shapiro, L. G., IEEE Trans. PAMI, Vol. PAMI-1. No. 2, April 1979, pp. 173-183.

[4] Montanari, U., Information Sciences 7 (1974), pp. 727-732

[5] Caulfield, H. J., Optics Communications, Vol. 55, No. 4, Dept., 1985, pp. 259-260.

U: {1,2,3,4,5} L: {a,b,c}

T: {(1,3),(1,4),(2,3),(3,4),(4,5),(1,5),(2,5)}



R:

(1a,3a)	(1a,4a)	(2a,3a)	(3a,4b)	(4a,5a)	(1b,5a)	(2a,5b)
(1a,3b)	(1a,4b)	(2c,3a)	(3b,4a)	(4b,5b)	(1a,5b)	(2c,5b)
(1b,3b)	(1b,4b)	(2c,3b)	(3b,4c)	(4a,5c)	(1c,5c)	(2c,5c)
(1c,3c)	(1c,4a)		(3c,4a)	(4c,5b)		(2b,5c)
(1c,3b)	(1c,4c)		(3c,4c)			(2b,5b)
	(1b,4c)					(2b,5a)

Figure 1. Constraint Relations: Example

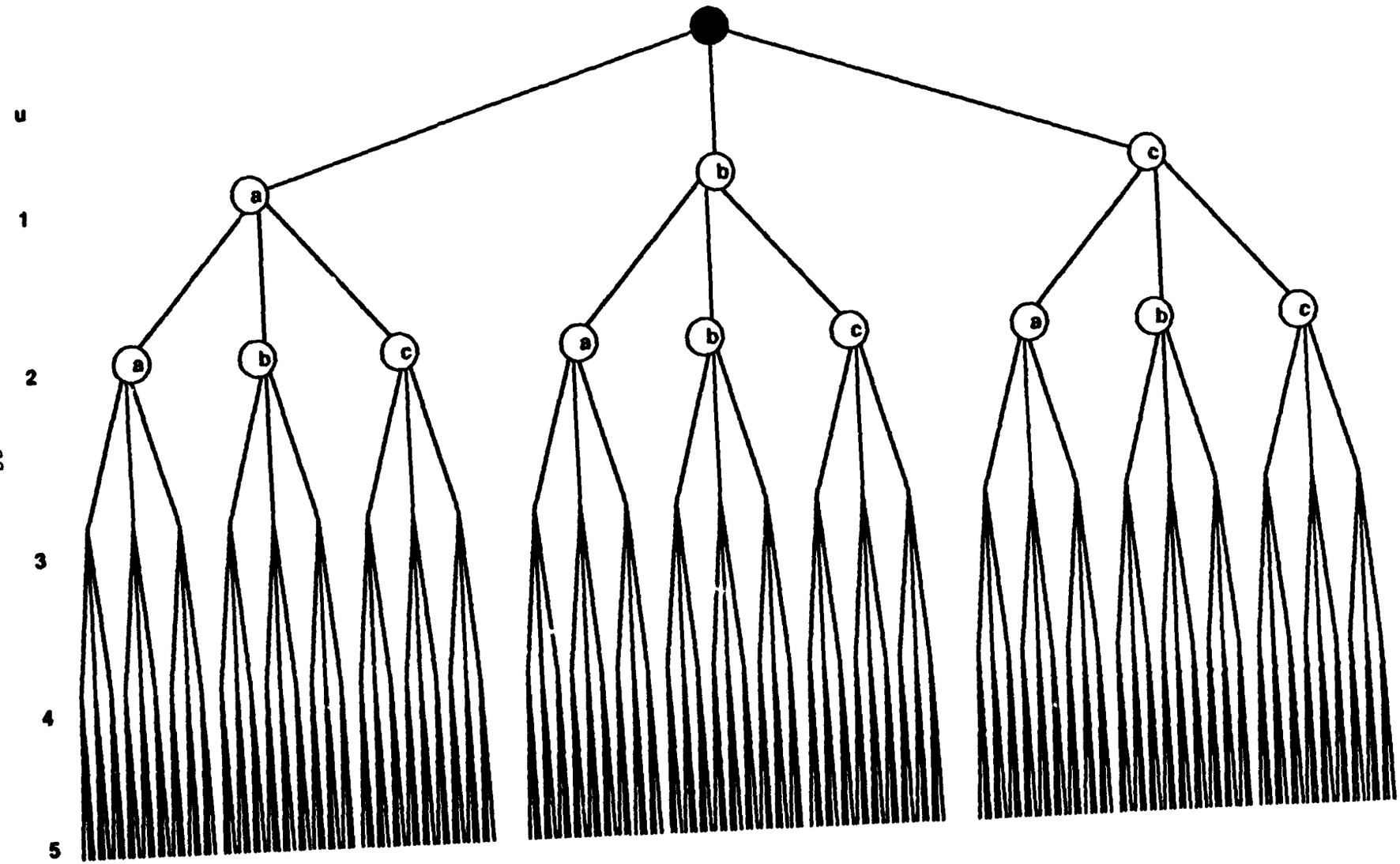


Figure 2. Initial Search Tree for Example Problem

$$R(1,3)' = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

$$R(1,4) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

$$R(2,3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

$$R(3,4) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

$$R(4,5) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$R(1,5) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R(2,5) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

Figure 3. Constraint Matrices for Example Problem

$$R(1,3)' = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$R(1,4) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

$$R(2,3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

$$R(3,4) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$R(4,5) = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$R(1,5) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R(2,5) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Figure 4. Constraint Matrices After Propagation of Initial Unary Constraints

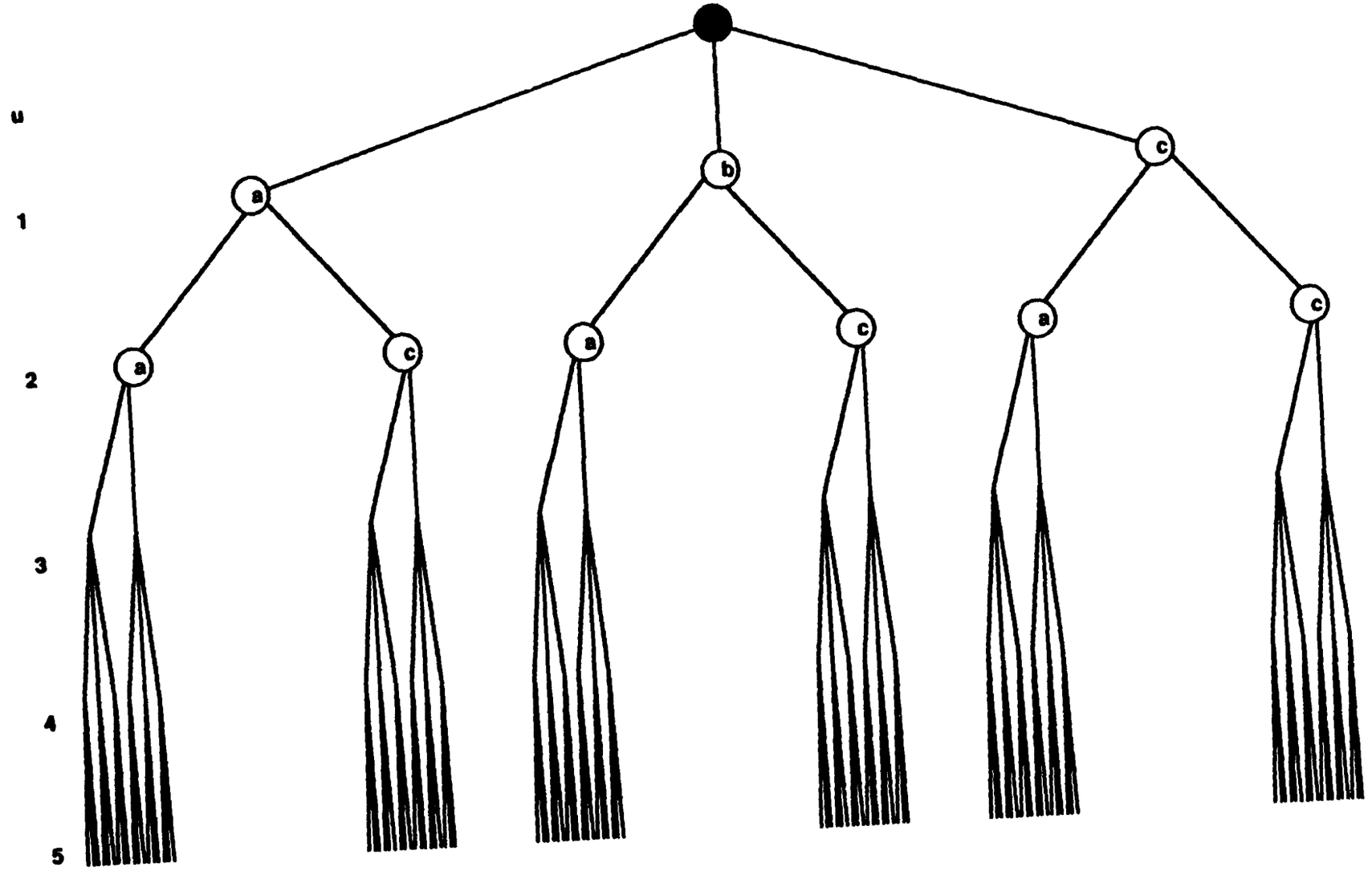


Figure 5. Search Tree After Computation of Arc Consistent Network

$$R(1,3)' = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$R(1,4) = \begin{pmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix}$$

$$R(2,3) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 1 & 1 & 0 \end{pmatrix}$$

$$R(3,4) = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

$$R(4,5) = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}$$

$$R(1,5) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

$$R(2,5) = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 1 & 1 \end{pmatrix}$$

Figure 6. Constraint Matrices After Forward-checking Some Paths of Length 2

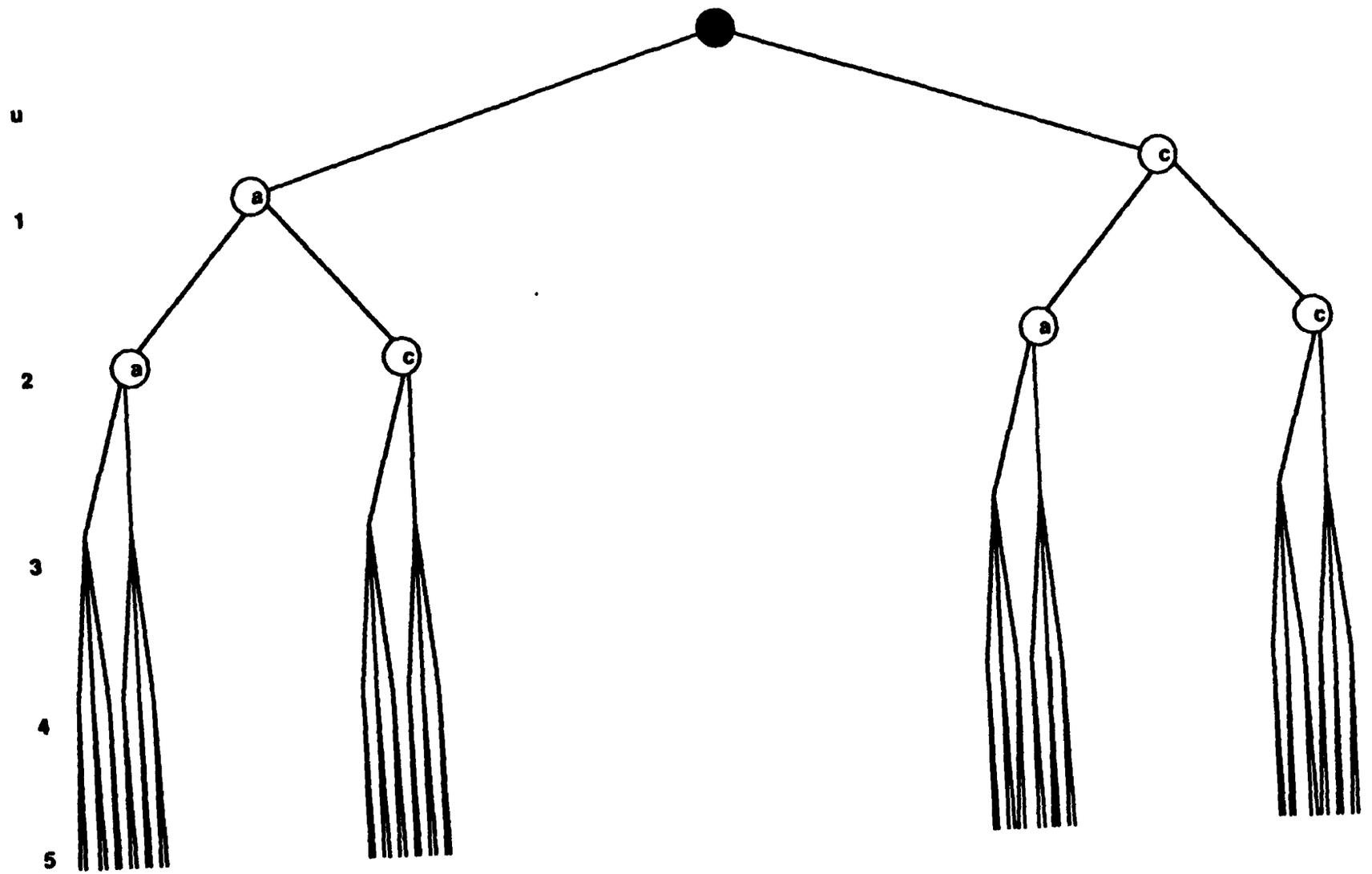


Figure 7. Search Tree After Forward-checking